International Journal of Computer Applications (0975 8887)
Volume 106 - No. 4, November 2014

Mapcode Theory in Designing Programs

Venkata Rao Kuchibhotla
Government College (A),
Rajahmundry,
A.P., India-533101

B.Gayatri Devi
Godavari Institute of Engineering and Technology,
Rajanagaram,
A.P., India-533296

ABSTRACT

A methodology is proposed for designing a program that takes care of safety (semantics) and termination using Mapcode theory.

General Terms:

Theory of Computation, program designing

Keywords:

partial recursive functions, primal maps, mapcode theory

1. INTRODUCTION

The concept of computability in the theory of computation is introduced to the students through many models like Turing machines, Partial Recursive Functions, λ-calculus, Post machines, and Register machines. When dealing with practical computing problems, the same students need to learn a programming language and study how to express algorithms in that language. Thus there is a disconnect between theory and practice.

In addition to the problem mentioned above, there are issues like analysis and design of programs which require a study on the principles involved in it. David Gries in his book [4] pointed out that

> "Programming began as an art, and even to day most people learn only by watching others perform (e.g. a lecturer, a friend) and through habit, with little direction as to the principles involved. ... we are reaching the point where we can begin to teach the principles so that they can be consciously applied".

In a series of articles [9, 10, 11, 12, 13] and a book [14] a theory called 'mapcode formalism' has been presented as an alternative approach so that a student can learn both the theoretical and practical issues in terms of a single framework. In the article [7] a class of maps called primal maps using mapcode theory was introduced and it was proved that the class of primal maps coincide with the class of partial recursive maps.

In continuation of the work, in the article [8] a mathematical framework was proposed that systematically helps in analyzing programs, particularly when dealing with ideas as in [2]. The present article addresses some of the issues related to design of programs. Certain structures are identified in designing a program and certain classes of primal maps are identified based on the programing structure for computing them. It is proposed that the identified classes of maps act as building blocks in designing a program for computing a primal map. This is verified with some of the standard examples discussed in [5] which are used to train students in the discipline of programming. However, it is not claimed that the classification is complete. But this kind of study helps in understanding design of algorithms and ensuring safety and termination simultaneously. This article presents a class of primal maps called *maps admitting variations* and a program structure which works for any map in the class.

Let $\mathbb{N}$ and $\mathbb{P}$ denote the set of all non-negative integers and the set of all positive integers respectively. Maps under consideration are those maps which are defined on subsets of $\mathbb{N}^p$ into $\mathbb{N}^r$ for $p, r \in \mathbb{P}$. A function is a map from a subset of $\mathbb{N}^p$ into $\mathbb{N}$. A map f with domain $D \subseteq \mathbb{N}^p$ and codomain $\mathbb{N}^r$ is called a partial map. It will be denoted by either $f : \mathbb{N}^p \to \mathbb{N}^r$ or $f : D \subseteq \mathbb{N}^p \to \mathbb{N}^r$. The domain D of f can also be written as D_f or $dom(f)$ when it is necessary to indicate the dependence of D on f. If $D_f = \mathbb{N}^p$, f is said to be *total*. The symbol $\doteq$ may be read as 'is defined as'. The symbols S, X, T are used for specific purposes. They are subsets of $\mathbb{N}^n$ possibly for different values of n.

2. MAPCODE FORMALISM

The definition of algorithm and other concepts given below are particular cases of a more general setup presented in [14]. This article is confined to the subsets of $\mathbb{N}^q$ for some $q \in \mathbb{P}$.

DEFINITION 1. A map $F : D \subseteq \mathbb{N}^q \to \mathbb{N}^q$ is called a Program Map on $\mathbb{N}^q$. Let F be a program map on $\mathbb{N}^q$.

(1) $x \in D$ is called a *fixed point* of F if $F(x) = x$. The set of all fixed points of F is denoted by $fix(F)$. It is called the fixed point set of F.
(2) If $x \in D$ and $F(x) \in D$, then $F^2(x) \doteq F(F(x))$. More generally, if $x, F(x), \cdots, F^n(x)$ are all in D then $F^{n+1}(x) \doteq F(F^n(x))$. By convention, $F^0(x) = x$ for any x, and $F^1(x) = F(x)$.
(3) $con(F) \doteq \{x \in D \mid F^n(x) \in fix(F) \text{ for some } n \geq 0\}$ is called the *convergent set* of F, and elements of $con(F)$ are called *convergent points* of F.
(4) For any $x \in con(F)$, let $N(x)$ be the least $n \geq 0$ such that $F^n(x) \in fix(F)$. $N(x)$ is called the *runtime* of F at x.
(5) The map $F^\infty : con(F) \to fix(F)$ defined by $F^\infty(x) = F^{N(x)}(x)$ is called the *limit map* of F. □

REMARK 2. If $F = Id : \mathbb{N}^q \to \mathbb{N}^q$ is the identity map then $con(F) = fix(F) = \mathbb{N}^q$ and $F^\infty = Id$. Further, the runtime $N(x) = 0$ for every $x \in \mathbb{N}^q$. □

One can think of F as defining a computation. Knuth [6] says the same thing when he observes that a mathematical theory of algorithms can be built by modeling an algorithm as a map F. Given a state $x \in \mathbb{N}^q$ the computation proceeds by evaluating in turn $F(x), F^2(x), \cdots$. If $x \in con(F)$ then there comes a stage when $F^n(x) \in fix(F)$. The computation is then considered to terminate.

DEFINITION 3. (1) Let $\rho : S \subseteq \mathbb{N}^p \to \mathbb{N}^q$, $F : X \subseteq \mathbb{N}^q \to \mathbb{N}^q$, and $\pi : H \subseteq \mathbb{N}^q \to \mathbb{N}^r$ be partial maps. Then the ordered triple $M = (\rho, F, \pi)$ is called a *mapcode machine* (over $\mathbb{N}$). The maps ρ, F, and π are called the *coding map, program map*, and *decoding map* of the machine respectively.

(2) The set $\Omega \doteq \{s \in S \mid \rho(s) \in con(F), F^\infty(\rho(s)) \in H\}$ is called the *halting set* of the mapcode machine M.

(3) The map $\phi = \pi \circ F^\infty \circ \rho : \Omega \subseteq \mathbb{N}^p \to \mathbb{N}^r$ is called the *semantic map* of the mapcode machine M.

(4) Let $f : \mathbb{N}^p \to \mathbb{N}^r$ be a partial map with its domain D_f, and let $M = (\rho, F, \pi)$ be a mapcode machine. M *computes* f or M is an *algorithm* for computing f, if $D_f = \Omega$, the halting set of M and $f(s) = \phi(s)$ for every $s \in D_f$. □

The terminology of 'halting set' and the notations for halting set and semantic map are borrowed from [1]. However, this terminology in the mapcode theory [14] is more general than theirs.

The components of a mapcode machine, namely the maps ρ, F, and π, constitute its syntax. The semantic map ϕ gives the semantics of the machine.

3. SAFETY AND PROGRESS THEOREMS

Given a mapcode machine (ρ, F, π), to prove that a state x is convergent and the output is the specified one, there is a need to have some way of showing safety and progress. The concept of invariant principle ensures safety and the concept of a bound function [14] is useful for verifying the progress.

DEFINITION 4. Let $f : D_f \subseteq \mathbb{N}^p \to \mathbb{N}^r$ be a partial map and $M = (\rho, F, \pi)$ be a mapcode machine with $\rho : S \subseteq \mathbb{N}^p \to \mathbb{N}^q$, $F : X \subseteq \mathbb{N}^q \to \mathbb{N}^q$, and $\pi : H \subseteq \mathbb{N}^q \to \mathbb{N}^r$. Suppose that $D_f \subseteq S$.

(1) A nonempty subset $V \subseteq X$ is said to be *invariant* under F if $F(V) \subseteq V$.

(2) A subset $V \subseteq X$ is called an *invariant principle* for f at $s \in D_f$, if V is invariant under F, $\rho(s) \in V$, $V \cap fix(F) \subseteq H$ and $\pi(V \cap fix(F)) = \{f(s)\}$.

(3) A map $\lambda : X \to \mathbb{N}$ is called a *bound function* for F if for every $x \in X$ with $F(x) \in X$ either $x \in fix(F)$ or $\lambda(F(x)) < \lambda(x)$.

(4) For any $s \in S$ define $V_s = \{F^n(\rho(s)) \mid n \geq 0\}$. □

REMARK 5. (1) If $\rho(s) \notin X$ then F is not defined on $\rho(s)$. Then $V_s = \{\rho(s)\}$ and V_s is not invariant under F. If $\rho(s) \in X$, but $F(\rho(s)) \notin X$ then F is not defined on $F(\rho(s))$. Then $V_s = \{\rho(s), F(\rho(s))\}$ and V_s is not invariant under F. Continuing the argument inductively, V_s is invariant under F if and only if $F^n(\rho(s)) \in X$ for every $n \geq 0$.

(2) Let $x \in X$ and let $\lambda(x) = 0$. Then either $x \in fix(F)$ or $F(x) \notin X$. □

THEOREM 6. *Let $f : D_f \subseteq \mathbb{N}^p \to \mathbb{N}^r$ be a partial map, $M = (\rho, F, \pi)$ a mapcode machine and $D_f \subseteq S$. If λ is a bound function for F and if V_s is an invariant principle for f at every $s \in D_f$ then $D_f \subseteq \Omega$ and $\phi(s) = f(s)$ for every $s \in D_f$. Further, if '$F(V_s) \subseteq V_s \Rightarrow s \in D_f$' holds, then M is an algorithm for f.*

PROOF. Take any $s \in D_f$. Since V_s is invariant under F by the remark [5], $F^n(\rho(s)) \in X$ for every $n \in \mathbb{N}$. Let $\lambda(\rho(s)) = n$. If $F^n(\rho(s)) \notin fix(F)$ then by the remark [5], $\lambda(F^n(\rho(s))) > 0$. But on the other hand,

$$\begin{aligned} \lambda(F(\rho(s))) < \lambda(\rho(s)) = n &\Rightarrow \lambda(F(\rho(s))) \leq n-1, \\ \lambda(F^2(\rho(s))) < \lambda(F(\rho(s))) &\Rightarrow \lambda(F^2(\rho(s))) \leq n-2, \\ &\cdots \quad \cdots \\ \text{and finally } \lambda(F^n(\rho(s))) \leq 0, &\quad \text{which is a contradiction.} \end{aligned}$$

It follows that $F^n(\rho(s)) \in fix(F)$. Since V_s is an invariant principle, $F^n(\rho(s)) \in fix(F) \cap V_s \subseteq H$ and $\pi(F^n(\rho(s))) = f(s)$. Then $s \in \Omega$ and $\phi(s) = f(s)$. This proves that $D_f \subseteq \Omega$ and $\phi(s) = f(s)$ for $s \in D_f$.

Further suppose that '$F(V_s) \subseteq V_s \Rightarrow s \in D_f$'. Take any $s \in \Omega$. Then $F^k(\rho(s)) \in fix(F)$ for some k, so that $F^n(\rho(s))$ is defined for all n. Then $F(V_s) \subseteq V_s$. By hypothesis, $s \in D_f$. This proves that $\Omega \subseteq D_f$. Hence $D_f = \Omega$ and M is an algorithm for f. □

REMARK 7. The concept of a bound function can be generalized to a map $\lambda : X \to P$ where P is a partial order set. Let P be a partial order set such that there is no sequence $\{a_n\}$ in P with $a_n > a_{n+1}$ for every n. A map $\lambda : X \to P$ is called a bound function for F if for every $x \in X$ with $F(x) \in X$ either $x \in fix(F)$ or $\lambda(F(x)) < \lambda(x)$. With this concept, if λ is a bound function and if V_s is an invariant principle for f at every $s \in D_f$ then $D_f \subseteq \Omega$ and $f(s) = \phi(s)$ for every $s \in D_f$. Further, if '$F(V_s) \subseteq V_s \Rightarrow s \in D_f$' then M is an algorithm for f. □

4. PRIMAL MAPS

In this section, a collection of maps called primal maps is introduced. It is a sub collection of the set of all maps from $\mathbb{N}^p$ to $\mathbb{N}^r$, with p and r varying in $\mathbb{P}$.

DEFINITION 8. (1) Let $f : D_f \subseteq \mathbb{N}^p \to \mathbb{N}^q$ and $g : D_g \subseteq \mathbb{N}^q \to \mathbb{N}^r$. Define $D = \{x \in D_f \mid f(x) \in D_g\}$. Then the *composition* $g \circ f : D \subseteq \mathbb{N}^p \to \mathbb{N}^r$ is given by $(g \circ f)(x) = g(f(x))$.

(2) If $f : D_f \subseteq \mathbb{N}^p \to \mathbb{N}^q$ and $g : D_g \subseteq \mathbb{N}^p \to \mathbb{N}^r$ are two maps then their *juxtaposition* $(f, g) : D_f \cap D_g \subseteq \mathbb{N}^p \to \mathbb{N}^{q+r}$ is defined by $(f, g)(x) = (f(x), g(x))$.

(3) Let $f_i : D_{f_i} \subseteq \mathbb{N}^p \to \mathbb{N}^q$, $g_i : D_{g_i} \subseteq \mathbb{N}^p \to \mathbb{N}^r$ for $i = 1, 2$. The conditional map $(f_1, f_2)|(g_1, g_2)$ is defined by

$$(f_1, f_2)|(g_1, g_2)(x) = \begin{cases} f_1(x) & \text{if } g_1(x) = g_2(x); \\ f_2(x) & \text{if } g_1(x) \neq g_2(x). \end{cases}$$

The sets

$$\begin{aligned} D_{g_1 = g_2} &\doteq \{x \in D_{g_1} \cap D_{g_2} \mid g_1(x) = g_2(x)\}, \text{ and} \\ D_{g_1 \neq g_2} &\doteq \{x \in D_{g_1} \cap D_{g_2} \mid g_1(x) \neq g_2(x)\} \end{aligned}$$

are called *guards* of the conditional map and the maps f_1 and f_2 are called *command maps*. The maps f_1, f_2, g_1 and g_2 are called *components* of the conditional map. The map $(f_1, f_2)|(g_1, g_2)$ is said to be obtained by *conditioning* (f_1, f_2) with respect to (g_1, g_2). The domain of the conditional map is the set $(D_{f_1} \cap D_{g_1 = g_2}) \cup (D_{f_2} \cap D_{g_1 \neq g_2})$.

(4) (a) $zero : \mathbb{N} \to \mathbb{N}$ is defined by $zero(x) = 0$ for all $x \in \mathbb{N}$.
(b) $succ : \mathbb{N} \to \mathbb{N}$ is defined by $succ(x) = x + 1$ for all $x \in \mathbb{N}$.
(c) For $0 \leq i < m$ the projection map $P_i^m : \mathbb{N}^m \to \mathbb{N}$ is defined by $P_i^m(x) = x[i-1]$ for all $x = (x[0], x[1], \ldots, x[m-1]) \in \mathbb{N}^m$.
The functions $zero$, $succ$, and P_i^m are called initial functions.

(5) For any set A define a function χ_A by $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ if $x \notin A$. The function χ_A is called indicator function of A. □

DEFINITION 9. The class of *primal maps* is the smallest class of maps that contains the initial functions and is closed under juxtaposition, conditional maps and semantic maps of mapcode machines (ρ, F, π).
A set is said to be *primal* if its indicator function is primal. □

REMARK 10. *The term 'closed under an operation' needs some explanation. It is explained in the case of semantic maps, and argument is similar for the other cases. A class $\mathcal{C}$ is closed under semantic maps of mapcode machines if $\phi = \pi \circ F^\infty \circ \rho \in \mathcal{C}$ whenever $\rho, F, \pi \in \mathcal{C}$.* □

It is proved in [7] that the class of primal maps is the class of partial recursive maps from the classical theory of computation [3]. This shows that the set of operations involved in generating partial recursive maps from initial maps can be replaced by the set of operations involved in generating primal maps.

A class of maps called "maps admitting variations" is introduced in the next section and a design for computing such maps is proposed.

5. MAPS ADMITTING VARIATIONS

For any $x = (x[0], x[1], \ldots, x[k-1]) \in \mathbb{N}^k$ and $y \in \mathbb{N}$, the juxtaposition of x, y is denoted by (x, y) so that $(x, y) = (x[0], x[1], \ldots, x[k-1], x[k]) \in \mathbb{N}^{k+1}$ with $x[k] = y$.

DEFINITION 11. (1) Let $f_1 : D_1 \subseteq \mathbb{N}^k \to \mathbb{N}^r$ and $f_2 : D_2 \subseteq \mathbb{N}^{k+1} \to \mathbb{N}^r$. The map f_1 is called a *variation* of f_2 if there exists a map $L : D_3 \subseteq \mathbb{N}^{k+1} \times \mathbb{N}^r \to \mathbb{N}^r$ as a nontrivial solution of the equation $f_2(x, y) = L(x, y, f_1(x))$, for any $(x, y) \in D_1$. The map L is called *lifting* of f_1 to f_2 (or simply a *lift map*). The equation $f_2(x, y) = L(x, y, f_1(x))$ is referred as *lift equation*.

(2) Let $p < n$, and let $f_k : D_k \subseteq \mathbb{N}^k \to \mathbb{N}^r$ for $p \leq k \leq n$ be given. If f_k is a variation of f_{k+1} for $p \leq k \leq n-1$ then the maps $f_p, f_{p+1}, \ldots, f_{n-1}$ are called *variations* of f_n. □

REMARK 12. (1) The map $L(x, y, t) = f_2(x, y)$ for every t, is always a solution for the lift equation and it is called a trivial solution.

(2) Suppose that $f_p, f_{p+1}, \ldots, f_{n-1}$ are variations of f_n. Then $(x_1, x_2, \ldots, x_n) \in dom(f_n) \Leftrightarrow (x_1, x_2, \ldots, x_k) \in dom(f_k)$ for $p \leq k \leq n$.

(3) More general concept of variations of a map f is given as a collection of maps f_i indexed by elements i of a partial order set P. In this case, f_i is a variation of f_j whenever j is a successor of i in P. □

EXAMPLE 1. Finding maximum (or minimum) of n integers and sum of n integers are simple examples to understand the idea behind the definition of variations of a map.

(1) **Maximum Problem:** For any $1 \leq k \leq n$, and any $x = (x[0], x[1], \ldots, x[k-1]) \in \mathbb{N}^k$, define $f_k(x) = Max\{x[0], x[1], \ldots, x[k-1]\}$. Then $f_k : \mathbb{N}^k \to \mathbb{N}$. It is easy to see that $Max\{x[0], x[1], \ldots, x[k]\} = Max\{x[k], Max\{x[0], x[1], \ldots, x[k-1]\}\}$ or $f_{k+1}(x, y) = Max\{y, f_k(x)\}$, for $x \in \mathbb{N}^k, y \in \mathbb{N}$.
More precisely, if $L_{k+1} : \mathbb{N}^{k+1} \times \mathbb{N} \to \mathbb{N}$ is defined by $L_{k+1}(x, y, t) = Max\{y, t\}$ for any $x \in \mathbb{N}^k, y, t \in \mathbb{N}$, then $f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$. The map L_{k+1} is lifting of f_k to f_{k+1}. The maps $f_1, f_2, \ldots, f_{n-1}$ are variations of $f = f_n$.

(2) **Summation Problem:** For any $1 \leq k \leq n$, and any $x = (x[0], x[1], \ldots, x[k-1]) \in \mathbb{N}^k$, define $f_k(x) = x[0] + x[1] + \ldots + x[k-1]$. Then $f_k : \mathbb{N}^k \to \mathbb{N}$. It is easy to see that $x[0] + x[1] + \ldots + x[k] = x[k] + (x[0] + x[1] + \ldots + x[k-1])$ or $f_{k+1}(x, y) = y + f_k(x)$, for $x \in \mathbb{N}^k, y \in \mathbb{N}$.
If $L_{k+1} : \mathbb{N}^{k+1} \times \mathbb{N} \to \mathbb{N}$ is defined by $L_{k+1}(x, y, t) = y + t$, for any $x \in \mathbb{N}^k, y, t \in \mathbb{N}$, then $f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$. Then the maps $f_1, f_2, \ldots, f_{n-1}$ are variations of $f = f_n$. □

Two more standard examples in the discipline of programming are given to fix the idea of variation in mind.

EXAMPLE 2. **Checking Equality Problem:** Choose and fix an element $x' = (x'[0], x'[1], \ldots, x'[p-1])$ in $\mathbb{N}^p$. For $k \in \mathbb{P}$ with $1 \leq k \leq p$, and $x = (x[0], \ldots, x[k-1]) \in \mathbb{N}^k$, define

$$\begin{aligned} f_k(x) &= 1, \text{ if } (x[0], \ldots, x[k-1]) = (x'[0], \ldots, x'[k-1]) \\ &= 0, \text{ if } (x[0], \ldots, x[k-1]) \neq (x'[0], \ldots, x'[k-1]). \end{aligned}$$

The maps $\{f_1, f_2, \ldots, f_{p-1}\}$ are variations of $f = f_p$. If $L_{k+1} : \mathbb{N}^{k+1} \times \mathbb{N} \to \mathbb{N}$ is defined by

$$\begin{aligned} L_{k+1}(x, y, t) &= t, \quad \text{if } y = s'[k], \text{ and} \\ L_{k+1}(x, y, t) &= 0, \quad y \neq s'[k], \end{aligned}$$

then $f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$. The map L_{k+1} is lifting of f_k to f_{k+1}. □

EXAMPLE 3. **Pattern Matching Problem**: The question is : how many times the pattern $s' = (s'[0],\ s'[1],\ \ldots\ s'[p-1])$ occur in the sequence $s = (s[0],\ s[1],\ \ldots\ s[n-1])$, assuming that n is many times larger than p.

Consider the notation and terminology used in the example [2]. For any $k \in \mathbb{P}$ with $p \leq k \leq n$, define $f_k : \mathbb{N}^k \to \mathbb{N}$ by $f_k(x)$ - the number of times the pattern x' occurs in x. The maps $\{f_p, f_{p+1}, \ldots, f_{n-1}\}$ are variations of f_n.

To find L_{k+1} − lifting of f_k to f_{k+1}, the map ψ_k which picks up the last p entries of the array $(s[0], s[1], \ldots, s[k-1])$ is needed. For any $p \leq k \leq n$, define the map $\psi_k : \mathbb{N}^k \to \mathbb{N}^p$ by $\psi_k(x[0], x[1], \ldots, x[k-1]) = (x[k-p], x[k-p+1], \ldots, x[k-1])$. Then for any $x = (x[0], x[1], \ldots, x[k]) \in \mathbb{N}^{k+1}$,

$$\begin{aligned} f_{k+1}(x) &= f_k(x[0], x[1], \ldots, x[k-1]) + 1, \text{ if } \psi_{k+1}(x) = x', \\ f_{k+1}(x) &= f_k(x[0], x[1], \ldots, x[k-1]), \text{ if } \psi_{k+1}(x) \neq x'. \end{aligned}$$

So, $f_{k+1}(x) = f_k(x[0], x[1], \ldots, x[k-1]) + f_p(\psi_{k+1}(x))$.
It is now easy to understand that the map $L_{k+1}(x, y, t)$ defined by $L_{k+1}(x, y, t) = t + f_p \circ \psi_{k+1}(x, y)$ is lifting of f_k to f_{k+1}. □

The purpose of considering variations of f is to split the computation of f into finite number of simple segments. The following theorem helps reaching this goal.

THEOREM 13. *suppose that $\{f_p, f_{p+1}, \ldots, f_{n-1}\}$ are variations of $f = f_n$. Let L_{k+1} be a lifting of f_k to f_{k+1}. If f_p and the map L_k are primal maps then f is a primal map.*

PROOF. Suppose that $f_k : \mathbb{N}^k \to \mathbb{N}^r$ for $p \leq k \leq n$. Given that $f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$ for $p \leq k \leq n-1$. Let $\eta_k : \mathbb{N}^n \to \mathbb{N}^k$ be defined by $\eta_k(x) = (x[0], x[1], \ldots, x[k-1])$ for $x = (x[0], x[1], \ldots, x[n-1]) \in \mathbb{N}^n$. Then the maps η_k are primal maps.

With $s = (s[0], s[1], \ldots, s[n-1]) \in \mathbb{N}^n$, $k \in \mathbb{N}$ and $t \in \mathbb{N}^r$, let $x = (s, k, t)$ be a general element of $\mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r$. Consider the partial maps $\rho : \mathbb{N}^n \to \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r$, $F : \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \to \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r$ and $\pi : \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \to \mathbb{N}^r$ given by

$$\begin{aligned} \rho(s) &= (s, p, f_p(\eta_p(s))), \\ F(s,k,t) &= \begin{cases} (s, k+1, L_{k+1}(\eta_{k+1}(s), t)\ ), \text{ if } k < n, \\ (s, k, t), \text{ otherwise} \end{cases} \\ \text{and } \pi(s,k,t) &= t. \end{aligned}$$

Then the maps ρ, F and π are all primal maps. Let the domains of the maps be S, X and H respectively. Then

$$\begin{aligned} S &= \{s \in \mathbb{N}^p \mid \eta_p(s) \in dom(f_p)\}, \\ X &= \{(s,k,t) \mid (\eta_{k+1}(s), t) \in dom(L_{k+1}), p \leq k \leq n-1\}, \\ H &= \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r. \end{aligned}$$

Clearly the map $\lambda(s, k, t) = n - k$ is a bound function. Further,

$$\begin{aligned} s \in \Omega &\Leftrightarrow \rho(s) \in X, F^k(\rho(s)) \in \mathit{fix}(F), \text{ for some } k \\ &\Leftrightarrow \begin{cases} (s, p, t_p) \in X, \text{ with } t_p = f_p \circ \eta_p(s) \\ (s, p+1, t_{p+1}) \in X, \text{ with } t_{p+1} = f_{p+1} \circ \eta_{p+1}(s) \\ \ldots \\ (s, n, t_n) \in X, \text{ with } t_n = f_n \circ \eta_n(s) \end{cases} \\ &\Leftrightarrow \begin{cases} \eta_p(s) \in dom(f_p), \\ \eta_{p+1}(s) \in dom(f_{p+1}) \\ \ldots \\ \eta_n(s) \in dom(f_n) \end{cases} \\ &\Leftrightarrow s \in dom(f_n), \text{ from remark[12].} \end{aligned}$$

This proves that $D_{f_n} = \Omega$. For any $s \in D_{f_n}$, V_s is an invariant set under F, and V_s can also be seen as

$$V_s = \{(s,k,t) \mid s \in D_{f_n}, 1 \leq k \leq n, t = f_k(\eta_k(s))\}$$

so that $V_s \cap \mathit{fix}(F) = \{(s, n, f_n(s))\}$, for $s \in D_{f_n}$. This proves that V_s is an invariant principle for f_n and (ρ, F, π) is an algorithm for computing the map f_n. Hence the map f_n is a primal map. □

6. DOCUMENTATION

This section deals with a method of documentation for the mapcode computation. At the outset, the mapcode machine

$$\begin{aligned} \rho(s) &= (s, p, f_p \circ \eta_p(s)) \\ F(s,k,t) &= \begin{cases} (s, k+1, L_{k+1}(\eta_{k+1}(s), t)\ ), \text{ if } k < n, \\ (s, k, t), \text{ otherwise} \end{cases} \\ \text{and } \pi(s,k,t) &= t. \end{aligned}$$

itself is a documentation for computing f. But tabular form may be a better option. $a \leftarrow b$ stands for "a is replaced by b".

Mapcode Documentation

Variables required	$s[0], s[1], \ldots s[n-1]$, $k, t[0], t[1], \ldots, t[r-1]$
Initialisation	
$0 \leq i \leq n-1$	input for $s[i]$ $t = f_p(s[0], \ldots, s[p-1])$ $k = p$
Program	
Guard	**Command**
$k < n$	$t \leftarrow L_{k+1}(\eta_{k+1}(s), t)$,
	$k \leftarrow k+1$
Output	t

7. APPLICATION

The proposed computation model will be used in the case of examples discussed earlier. The advantage of this theory is that the safety and termination are ensured by the theorem and there is no need to verify for each example.

(1) **Maximum Problem:** With the notation and terminology given in the example 1,
   (a) the map f_1 is identity map on $\mathbb{N}$, so that it is a primal map.
   (b) The lift map $L_k(x, y, t) = Max\{y, t\}$ is a primal map.

   Then the maximum function f is a primal map and the mapcode machine computing f is given by

$$\begin{aligned} \rho(s) &= (s, 1, s[0]) \\ F(s,k,t) &= \begin{cases} (s, k+1, Max\{s[k], t\}), \text{ if } k < n, \\ (s, k, t), \text{ otherwise }, \end{cases} \\ \text{and } \pi(s,k,t) &= t. \end{aligned}$$

   Documentation of the computation in tabular form is given by

Mapcode Documentation

Variables required	$s[0], s[1], \ldots s[n-1]$, k, t
Initialisation	
$0 \leq i \leq n-1$	input for $s[i]$ $t = s[0]$ $k = 1$
Program	
Guard	**Command**
$k < n$	$t \leftarrow Max\{s[k], t\}$,
	$k \leftarrow k+1$
Output	t

(2) **Checking Equality Problem:** With the terminology and notation given in the example [2]
   (a) $s' = (s'[0], s'[1], \ldots, s'[p-1]) \in \mathbb{N}^p$ is initially choosen and fixed.
   (b) The map $f_1 : \mathbb{N} \to \mathbb{N}$ is given by $f_1(x) = 1$, if $x = s'[0]$, and $f_1(x) = 0$, otherwise. It is a primal map.
   (c) For any $(s, y) \in \mathbb{N}^{k-1} \times \mathbb{N} = \mathbb{N}^k$ and $t \in \mathbb{N}$, the lift map L_k is given by $L_k(s, y, t) = t$, if $y = s'[k]$, and $L_k(s, y, t) = 0$, otherwise. Then L_k is also a primal map.

   Then by theorem 13, the function $f = f_p$ which checks equality is a primal map. The mapcode machine computing f is given by

$$\begin{aligned}\rho(s,s') &= (s,s',1,f_1(s))\\ F(s,s',k,t) &= (s,s',k+1,L_k(s[k],t)),\ \text{if } k<p\\ &= (s,s',k,t),\ \text{otherwise and}\\ \pi(s,s',k,t) &= t.\end{aligned}$$

In other words,

$$\begin{aligned}\rho(s,s') &= \begin{cases}(s,s',1,1),\ \text{if } s[0]=s'[0]\\ (s,s',1,0),\ \text{if } s[0]\neq s'[0]\end{cases}\\ F(s,s',k,t) &= \begin{cases}(s,s',k+1,t), \text{if } s[k]=s'[k], \text{and } k<p\\ (s,s',k+1,0), \text{if } s[k]\neq s'[k], \text{and } k<p\end{cases}\\ &= (s,s',k,t),\ \text{otherwise and}\\ \pi(s,s',k,t) &= t.\end{aligned}$$

However, a close observation of the machine will reveal that the following machine will also compute the map f.

$$\begin{aligned}\rho(s,s') &= (s,s',0,1)\\ F(s,s',k,t) &= \begin{cases}(s,s',k+1,t),\ \text{if } k<p, s[k]=s'[k]\\ (s,s',n,0),\ \text{if } k<p, s[k]\neq s'[k]\end{cases}\\ &= (s,s',k,t),\ \text{otherwise;}\\ \pi(s,s',k,t) &= t.\end{aligned}$$

Mapcode Documentation

Variables required	$s[0],\ldots s[n-1]$, $s'[0],\ldots s'[n-1]$, k,t
Initialisation	
$0\le i\le n-1$ $0\le i\le n-1$	input for $s[i]$ input for $s'[i]$ $k=0, t=1$
Program	
Guard	**Command**
$k<p, s[k]=s'[k]$	$k\leftarrow k+1$
$s[k]\neq s'[k]$	$k\leftarrow n, t\leftarrow 0$
Output	t

This mapcode machine is referred by M_1.

(3) **Pattern Matching Problem**: This famous problem has been tackled by many programmers. One can see how the proposed method simplifies the problem of designing a program. Assume the notation and terminology used in the example [3].

(a) Choose and fix any $s'=(s'[0],\ s'[1],\ \ldots s'[p-1])\in\mathbb{N}^p$.
(b) The map f_p in the example 3 is same as the map f_p in the example 2 and hence f_p is a primal map.
(c) The lift map L_k given by $L_k(s[0],s[1],\ldots,s[k-1],t)=t+f_p(s[k-p],s[k-p+1],\ldots,s[k-1])$ is also a primal map.

By theorem 13, the function $f=f_n$ is a primal map. The mapcode machine (ρ,F,π) computing f is given by

$$\begin{aligned}\rho(s,s') &= (s,s',p,f_p(s[0],\ldots,s[p-1])\ )\\ \pi(x) &= t.\end{aligned}$$

and for any $x=(s,s',k,t)\in X$,

$$\begin{aligned}F(x) &= (s,s',k+1,t+f_p(s[k+1-p],\ldots,s[k])\ ),\\ &\quad \text{if } k<n,\\ &= (s,s',k,t),\ \text{otherwise}\end{aligned}$$

Mapcode Documentation

Variables required	$s[0],\ldots s[n-1]$, $s'[0],\ldots s'[n-1]$, k,t
Initialisation	
$0\le i\le n-1$ $0\le i\le n-1$	input for $s[i]$ input for $s'[i]$ $t=f_p(s[0],\ldots,s[p-1])$ $k=p$
Program	
Guard	**Command**
$k<n,$	$k\leftarrow k+1$ $t\leftarrow t+f_p(s[k+1-p],\ldots,s[k])\ )$
Output	t

This mapcode machine computing f_n is referred by M_2.

8. CONCLUSION

The mapcode machine M_2 involves maps which are again computable with the machine M_1. Then one has to combine the mapcode machines M_1 and M_2 so that relatively elementary operations will be involved for carrying out the computation. This is an essential part of writing a program.

As in the case of 'maps admitting variations', some other classes of maps are identified and proposed a mapcode machine for each class. Though the approach is successfully adopted for several standard problems, some of them are even harder, it is not claimed that these classes are exhaustive. These results and the techniques involved in combining two mapcode machines will be published subsequently.

In the present article, simple illustrations are chosen only for better exposition of the content. Two standard examples are chosen to show how the proposed method helps in the construction of an algorithm. However, as quoted by David Gries [4],

> One cannot learn to write large programs effectively
> until one has learned to write small ones effectively.

9. REFERENCES

[1] Blum, L., Cucker, F., Shub, M., and Smale, S.: *Complexity and Real Computation*, Springer-Verlag, New York, USA, 1998.

[2] Chandy, K.M., Misra, J.: *Parallel Program Design: A Foundation,* Addison-Wesley, Reading, 1988.

[3] Cutland, N.: *Computability:An Introduction to Recursive Function Theory,* Cambridge University Press, Cambridge, UK, 1980.

[4] David Gries.: *The Scinece of Programming* Springer-Verlag New York Inc, 1981.

[5] Edsger W. Dijkstra.: *A Discipline of Programming* Prentice-Hall of India, New Delhi, India, 1979.

[6] Knuth, D.E.: *The Art of Computer Programming*, Volumes 1 - III, Third Edition, Pearson Education Asia, New Delhi, India, 2002.

[7] Venkata Rao, K., and Viswanath, K.: *Mapcode Characterization of Partial Recursive Maps,* Sankhya, The Indian Journal of Statistics, 2010, Volume 72-A, Part 1, pp. 276-292.

[8] Venkata Rao, K., and Viswanath, K.: *Computing with Multiple Discrete Flows*, Journal of Differential Equations and Dynamical Systems, 2010, Volume 18 (4), pp. 401-414.

[9] Viswanath, K: *Dynamical Systems and Computer Science*, Mathematics Newsletter of the Ramanujan Mathematical Society, 13(3), 2003, pp.33-48.

[10] Viswanath, K.: *Building Programs and proving them correct with Dynamical Systems*, Proceedings of the International Conference on Computing and Control, Austin, Texas, USA, August, 2004, International Institute of Informatics and Systemics,Orlando, Florida, USA, pp. 372-377.

[11] Viswanath, K.: *An Invitation to Mathematical Computer Science*, Discrete Mathematics and its Applications, Ed. M. Sethumadhavan, Narosa, 2006.

[12] Viswanath, K.: *Simple Foundations for Computing*, Proceedings of the 40th Annual Convention of the Computer Society of India, November 10-12, 2005, Hyderabad, India.

[13] Viswanath, K.: *Computing with Dynamical Systems*, Journal of Differential Equations and Dynamical Systems, Vol.14(1), pp.1-24, 2006.

[14] Viswanath, K.: *An Introduction to Mathematical Computer Science*, The Universities Press, Hyderabad, India, 2008.